NASA/CR-1999-209724
ICASE Report No. 99-44

DISTRIBUTION STATEMENT A
Approved for Public Release
Distribution Unlimited

Large-scale Parallel Viscous Flow Computations Using an Unstructured Multigrid Algorithm

Dimitri J. Mavriplis
ICASE, Hampton, Virginia

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA

Operated by Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NAS1-97046

November 1999

DTIC QUALITY INSPECTED 4

19991209 082

DISTRIBUTION STATEMENT AUTHORIZATION RECORD

Title: Large-scale Parallel Viscous Flow Computations Using an Unstructured Multigrid Algorithm

Authorizing Official: Dimitri Mavriplis

Agency: NASA Ph. No. DSN [redacted] 864-2213

☐ Internet Document: URL: _____
(DTIC-OCA Use Only)

Distribution Statement: (Authorized by the source above.)

☑ **A:** Approved for public release, distribution unlimited.

☐ **B:** U. S. Government agencies only. (Fill in reason and date applied). Other requests shall be referred to (Insert controlling office).

☐ **C:** U. S. Government agencies and their contractors. (Fill in reason and date applied). Other requests shall be referred to (Insert controlling office).

☐ **D:** DoD and DoD contractors only. (Fill in reason and date applied). Other requests shall be referred to (Insert controlling office).

☐ **E:** DoD components only. (Fill in reason and date applied). Other requests shall be referred to (Insert controlling office).

☐ **F:** Further dissemination only as directed by (Insert controlling DoD office and date), or higher authority.

☐ **X:** U. S. Government agencies and private individuals or enterprises eligible to obtain export-controlled technical data in accordance with DoD Directive 5230.25.

NOTES: _____

J. Kingsley
DTIC Point of Contact

1-5-2000
Date

LARGE-SCALE PARALLEL VISCOUS FLOW COMPUTATIONS USING AN UNSTRUCTURED MULTIGRID ALGORITHM

DIMITRI J. MAVRIPLIS*

Abstract. The development and testing of a parallel unstructured agglomeration multigrid algorithm for steady-state aerodynamic flows is discussed. The agglomeration multigrid strategy uses a graph algorithm to construct the coarse multigrid levels from the given fine grid, similar to an algebraic multigrid approach, but operates directly on the non-linear system using the FAS approach. The scalability and convergence rate of the multigrid algorithm are examined on the SGI Origin 2000 and the Cray T3E. An argument is given which indicates that the asymptotic scalability of the multigrid algorithm should be similar to that of its underlying single grid smoothing scheme. For medium size problems involving several million grid points, near perfect scalability is obtained for the single grid algorithm, while only a slight drop-off in parallel efficiency is observed for the multigrid V- and W-cycles, using up to 128 processors on the SGI Origin 2000, and up to 512 processors on the Cray T3E. For a large problem using 25 million grid points, good scalability is observed for the multigrid algorithm using up to 1450 processors on a Cray T3E, even when the coarsest grid level contains fewer points than the total number of processors.

Key words. multigrid, anisotropic, Navier-Stokes

Subject classification. Applied and Numerical Mathematics

1. Introduction. Reynolds averaged Navier-Stokes computations using several million grid points have become commonplace today. While many practical problems can be solved to acceptable accuracy with such methods at these resolutions, the drive to more complex problems and higher accuracy is requiring the solution of ever larger problems. For example, the flow over aircraft configurations in off-design configurations, such as high-lift, has been computed with up to 25 million grid points, and cases involving up to 10^8 grid points can be anticipated in the near future [14].

The elusive goal of developing a universally valid turbulence model has also spurred a new interest in large eddy simulation (LES) models, which may ultimately require in excess of 10^9 or even 10^{10} grid points for adequate resolution of the relevant eddy sizes, even excluding the thin boundary layer regions. At the same time, the drive towards more complex configurations and faster gridding turnaround time has emphasized the use of unstructured grid methods. While unstructured grid techniques simplify the task of discretizing complex geometries, and offer great potential for the use of adaptive meshing techniques, they incur additional cpu and memory overheads as compared to block-structured or overset grid methods. On the other hand, unstructured grid methods are well suited for large scale parallelization, since the basic data structures are homogeneous in nature, which enables near perfect load-balancing on very large numbers of processors.

Any large-scale solution procedure requires the use of an efficient solver, regardless of the amount of available computer resources. While a simple explicit scheme may achieve the best parallel efficiency on

*Institute for Computer Applications in Science and Engineering (ICASE), Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23681–2199, U.S.A., dimitri@icase.edu. This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-97046 while the author was in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681-2199.

large numbers of processors, its numerical efficiency (i.e. convergence rate) degrades rapidly as the number of grid points is increased. Multigrid algorithms have been shown to provide optimal asymptotic complexity, with the number of operations required for convergence scaling linearly with the number of grid points or unknowns. While other implicit solution techniques can provide fast convergence rates for large problems, FAS multigrid methods avoid the explicit linearization of the non-linear problem, resulting in an algorithm which requires considerably less storage, particularly for unstructured mesh discretizations. Furthermore, because multigrid methods are based on explicit or locally-implicit single grid smoothers, they are also highly memory-latency tolerant, an important consideration in light of current architectural trends, which has seen dramatic increases in the relative memory latency over the last several decades [8, 12]. Finally, as will be shown in this paper, multigrid methods can be expected to scale favorably for reasonable size problems on very large numbers of processors.

The use of cache-based microprocessor parallel computer architectures is rapidly becoming the dominant approach for large-scale CFD calculations. While parallel vector machines such as the Cray T90 and the NEC SX-4 offer outstanding performance as measured by computational rates, their required use of fast but costly memory has limited the amount of memory available on such machines. This is particularly important for unstructured grid computations, which have traditionally been memory limited. The use of low cost hierarchical (cache-based), high-latency memory systems is somewhat at odds with processors which rely on very long (global) vector lengths for sustaining high computational rates. Hence, the dramatically lower cost of commodity memory has made large scale parallel systems of cache-based microprocessors the most effective architecture for large unstructured grid computations.

Other enabling developments for parallel unstructured grid computations include the availability of efficient and robust grid partitioners [4, 7], and in particular the appearance of standardized software libraries for inter-processor communication such as the Message Passing Interface (MPI) library [3], which enable code portability and simplify maintenance.

In the following three sections, an unstructured mesh multigrid solver designed for turbulent external flow aerodynamic analysis is described. The convergence rate and scalability of this approach are illustrated by two examples in section 5, and a multigrid scalability argument is given in section 6. In section 7 the performance of calculations using up to 2048 processors and 25 million grid points is described, while section 8 discusses the outlook for even larger future calculations.

2. Base Solver. The Reynolds averaged Navier-Stokes equations are discretized by a finite-volume technique on meshes of mixed element types which may include tetrahedra, pyramids, prisms, and hexahedra. In general, prismatic elements are used in the boundary layer and wake regions, while tetrahedra are used in the regions of inviscid flow. All elements of the grid are handled by a single unifying edge-based data-structure in the flow solver [15].

The governing equations are discretized using a central difference finite-volume technique with added matrix-based artificial dissipation. The matrix dissipation approximates a Roe Rieman-solver based upwind scheme [19], but relies on a biharmonic operator to achieve second-order accuracy, rather than on a gradient-based extrapolation strategy [10]. The thin-layer form of the Navier-Stokes equations is employed in all cases, and the viscous terms are discretized to second-order accuracy by finite-difference approximation. For multigrid calculations, a first-order discretization is employed for the convective terms on the coarse grid levels.

The basic time-stepping scheme is a three-stage explicit multistage scheme with stage coefficients opti-

mized for high frequency damping properties [24], and a CFL number of 1.8. Convergence is accelerated by a local block Jacobi preconditioner, which involves inverting a 5×5 matrix for each vertex at each stage [16, 17]. A low-Mach number preconditioner [26, 22, 21] is also implemented in order to relieve the stiffness associated with the disparity in acoustic and convective eigenvalues in regions where the Mach number is very small and the flow behaves incompressibly. The low-Mach number preconditioner is implemented by modifying the dissipation terms in the residual as described in [10], and then taking the corresponding linearization of these modified terms into account in the Jacobi preconditioner, a process sometimes referred to as "*preconditioning*^{2}" [10, 23].

The single equation turbulence model of Spalart and Allmaras [20] is utilized to account for turbulence effects. This equation is discretized and solved in a manner completely analogous to the flow equations, with the exception that the convective terms are only discretized to first-order accuracy.

3. Directional-Implicit Multigrid Algorithm. An agglomeration multigrid algorithm [15, 9] is used to further enhance convergence to steady-state. In this approach, coarse levels are constructed by fusing together neighboring fine grid control volumes to form a smaller number of larger and more complex control volumes on the coarse grid. While agglomeration multigrid delivers very fast convergence rates for inviscid flow problems, the convergence obtained for viscous flow problems remains much slower, even when employing preconditioning techniques as described in the previous section. This slowdown is mainly due to the large degree of grid anisotropy in the viscous regions. Directional smoothing and coarsening techniques [10, 11] can be used to overcome this aspect-ratio induced stiffness.

Directional smoothing is achieved by constructing lines in the unstructured mesh along the direction of strong coupling (i.e., normal to the boundary layer) and solving the implicit system along these lines using a tridiagonal line solver. A weighted graph algorithm is used to construct the lines on each grid level, using edge weights based on the stencil coefficients for a scalar convection equation. This algorithm produces lines of variable length. In regions where the mesh becomes isotropic, the length of the lines reduces to zero (one vertex, zero edges), and the preconditioned explicit scheme described in the previous section is recovered. An example of the set of lines constructed from the two-dimensional unstructured grid in Figure 3.1 is depicted in Figure 3.2.

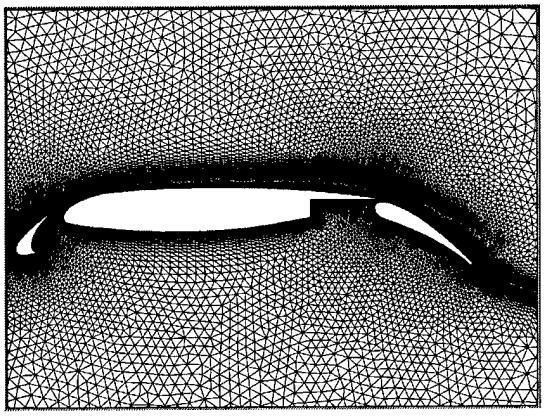

FIG. 3.1. *Unstructured Grid for Three-Element Airfoil; Number of Points = 61,104, Wall Resolution* $= 10^{-6}$ *chords*

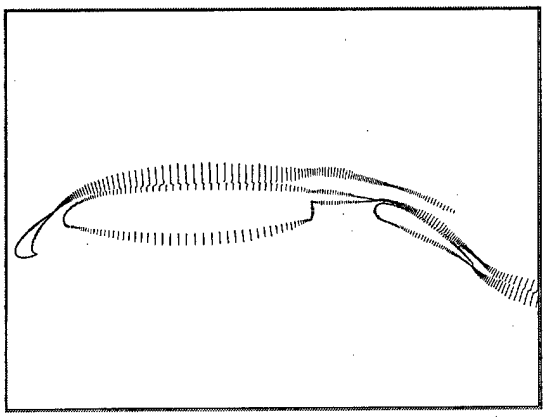

FIG. 3.2. *Directional Implicit Lines Constructed on Grid of Figure 3.1 by Weighted Graph Algorithm*

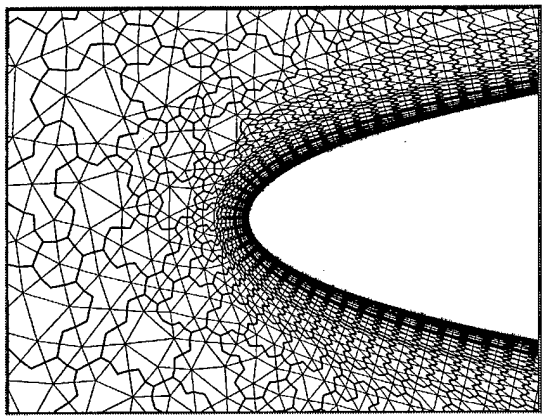

FIG. 3.3. *First Agglomerated Multigrid Level for Two-Dimensional Unstructured Grid Illustrating 4:1 Directional Coarsening in Boundary Layer Region*

In addition to using a directional smoother, the agglomeration multigrid algorithm must be modified to take into account the effect of mesh stretching. The unweighted agglomeration algorithm which groups together all neighboring control volumes for a given fine grid vertex [15] is replaced with a weighted coarsening algorithm which only agglomerates the neighboring control volumes which are the most strongly connected to the current fine grid control volume, as determined by the same edge weights used in the line construction algorithm.

This effectively results in semi-coarsening type behavior in regions of large mesh stretching, and regular coarsening in regions of isotropic mesh cells. In order to maintain favorable coarse grid complexity, an aggressive coarsening strategy is used in anisotropic regions, where for every retained coarse grid point, three fine grid control volumes are agglomerated, resulting in an overall complexity reduction of 4:1 for the coarser levels in these regions, rather than the 2:1 reduction typically observed for semi-coarsening techniques. In inviscid flow regions, the algorithm reverts to the isotropic agglomeration procedure and an 8:1 coarsening ratio is obtained. However, since most of the mesh points reside in the boundary layer regions, the overall coarsening ratios achieved between grid levels is only slightly higher than 4:1. An example of the first directionally agglomerated level on a two-dimensional mesh is depicted in Figure 3.3, where the aggressive agglomeration normal to the boundary layer is observed.

4. Parallel Implementation. Distributed-memory explicit message-passing parallel implementations of unstructured mesh solvers have been discussed extensively in the literature [13, 1, 25]. In this section we focus on the non-standard aspects of the present implementation which are particular to the directional-implicit agglomeration multigrid algorithm.

In the multigrid algorithm, the vertices on each grid level must be partitioned across the processors of the machine. Since the mesh levels of the agglomeration multigrid algorithm are fully nested, a partition of the fine grid could be used to infer a partition of all coarser grid levels. While this would minimize the communication in the inter-grid transfer routines, it affords little control over the quality of the coarse grid partitions. Since the amount of intra-grid computation on each level is much more important than the inter-grid computation between each level, it is essential to optimize the partitions on each grid level rather than between grid levels. Therefore, each grid level is partitioned independently. This results in unrelated coarse and fine grid partitions. In order to minimize inter-grid communication, the coarse level partitions are

renumbered such that they are assigned to the same processor as the fine grid partition with which they share the most overlap. For each partitioned level, the edges of the mesh which straddle two adjacent processors are assigned to one of the processors, and a "ghost vertex" is constructed in this processor, which corresponds to the vertex originally accessed by the edge in the adjacent processor (c.f. Figure 4.1). During a residual evaluation, the fluxes are computed along edges and accumulated to the vertices. The flux contributions accumulated at the ghost vertices must then be added to the flux contributions at their corresponding physical vertex locations in order to obtain the complete residual at these points. This phase incurs interprocessor communication. In an explicit (or point implicit) scheme, the updates at all points can then be computed without any interprocessor communication once the residuals at all points have been calculated. The newly updated values are then communicated to the ghost points, and the process is repeated.

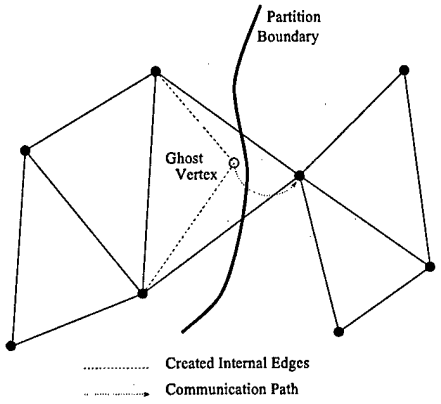

FIG. 4.1. *Illustration of Creation of Internal Edges and Ghost Points at Inter-processor Boundaries*

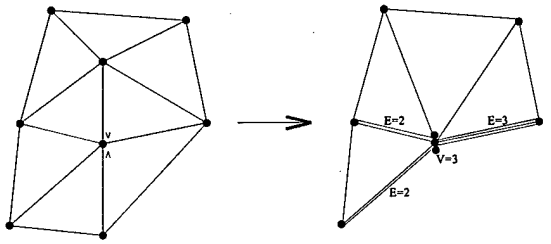

FIG. 4.2. *Illustration of Line Edge Contraction and Creation of Weighted Graph for Mesh Partitioning; V and E Values Denote Vertex and Edge Weights Respectively*

The use of line-solvers can lead to additional complications for distributed-memory parallel implementations. Since the classical tridiagonal line-solve is an inherently sequential operation, any line which is split between multiple processors will result in processors remaining idle while the off-processor portion of their line is computed on a neighboring processor. However, the particular topology of the line sets in the unstructured grid permit partitioning the mesh in such a manner that lines are completely contained within an individual processor, with minimal penalty (in terms of processor imbalance or additional numbers of cut edges). This can be achieved by using a weighted-graph-based mesh partitioner such as the CHACO [4] or METIS [6] partitioners. Weighted graph partitioning strategies attempt to generate balanced partitions of sets of weighted vertices, and to minimize the sum of weighted edges which are intersected by the partition boundaries. In order to avoid partitioning across implicit lines, the original unweighted graph (set of vertices and edges) which defines the unstructured mesh is contracted along the implicit lines to produce a weighted graph. Unity weights are assigned to the original graph, and any two vertices which are joined by an edge which is part of an implicit line are then merged together to form a new vertex. Merging vertices also produce merged edges as shown in Figure 4.2, and the weights associated with the merged vertices and edges are taken as the sum of the weights of the constituent vertices or edges. The contracted weighted graph is then partitioned using the partitioners described in [6, 5], and the resulting partitioned graph is then de-contracted, i.e., all constituent vertices of a merged vertex are assigned the partition number of that vertex. Since the implicit lines reduce to a single point in the contracted graph, they can never be broken by the partitioning process. The weighting assigned to the contracted graph ensures load balancing and

communication optimization of the final uncontracted graph in the partitioning process.

As an example, the two dimensional mesh in Figure 3.1, which contains the implicit lines depicted in Figure 3.2, has been partitioned both in its original unweighted uncontracted form, and by the graph contraction method described above. Figure 4.3 depicts the results of both approaches for a 32-way partition. The unweighted partition contains 4760 cut edges (2.6 % of total), of which 1041 are line edges (also 2.6 % of total), while the weighted partition contains no intersected line edges and a total of 5883 cut edges (3.2 % of total), i.e., a 23% increase over the total number of cut edges in the non-weighted partition.

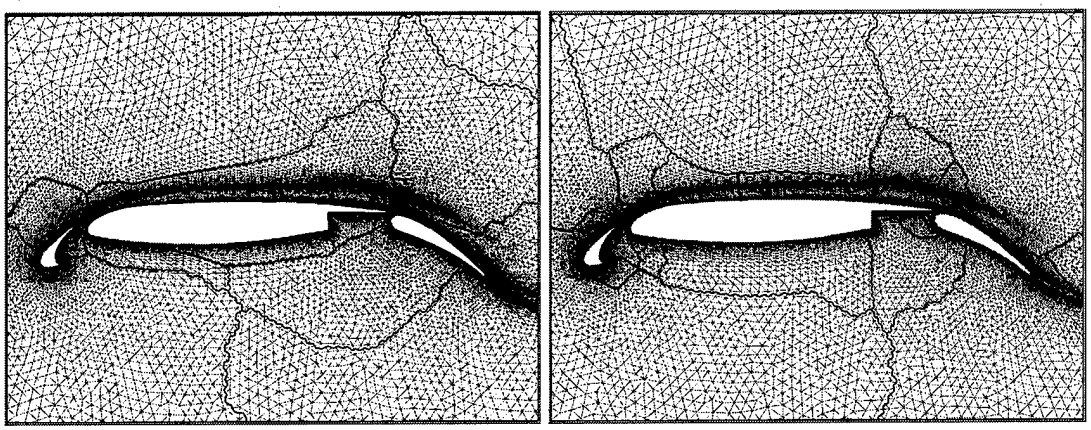

FIG. 4.3. *Comparison of Unweighted (left) and Weighted (right) 32-Way Partition of Two-Dimensional Mesh*

5. Scalability Results. Two test cases are employed to examine the convergence behavior and scalability of the directional implicit parallel unstructured multigrid solver. The first case consists of a relatively coarse 177,837 point grid over a swept and twisted wing, constructed by extruding a two-dimensional grid over an RAE 2822 airfoil in the spanwise direction. Figure 5.1 illustrates the grid for this case along with the implicit lines used by the solution algorithm on the finest level. The grid contains hexahedra in the boundary layer and (spanwise) prismatic elements in regions of inviscid flow, and exhibits a normal spacing at the wing surface of 10^{-6} chords. Approximately 67% of the fine grid points are contained within an implicit line, and no implicit lines on any grid levels were intersected in the partitioning process for all cases. This case was run at a freestream Mach number of 0.1, an incidence of 2.31 degrees, and a Reynolds number of 6.5 million. The convergence of the directional implicit multigrid algorithm is compared with that achieved by the explicit isotropic multigrid algorithm [15] on the equivalent two dimensional problem in Figure 5.2. The directional implicit multigrid algorithm is seen to be much more effective than the isotropic algorithm, reducing the residuals by twelve orders of magnitude over 600 multigrid W-cycles.

The second test case involves a finer grid of 1.98 million points over an ONERA M6 wing. The grid was generated using the VGRID unstructured tetrahedral mesh generation program [18]. A post-processing operation was employed to merge the tetrahedral elements in the boundary layer region into prisms [15, 14]. The final grid contains 2.4 million prismatic elements and 4.6 million tetrahedral elements, and exhibits a normal spacing at the wall of 10^{-7} chords. Approximately 62% of the fine grid points are contained within an implicit line, and no implicit lines on any grid levels were intersected in the partitioning process for all cases. The freestream Mach number is 0.1, the incidence is 2.0 degrees, and the Reynolds number is 3 million. The residuals are reduced by seven orders of magnitude over 600 multigrid W-cycles in this case.

This convergence rate is somewhat slower than that achieved on the previous problem, and than the rates obtained on two-dimensional problems using the same algorithm [11]. This is typical of the convergence rates obtained by the current algorithm on genuinely three-dimensional problems.

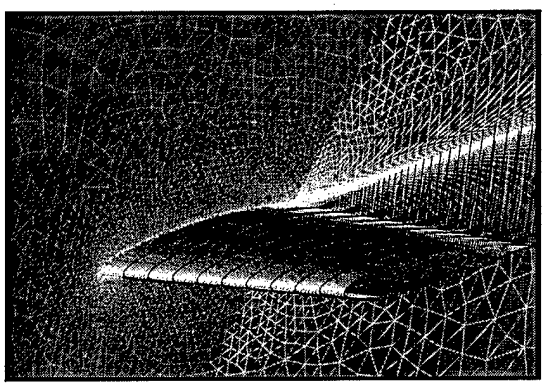

FIG. 5.1. *Unstructured Grid and Implicit Lines Employed for Computing Flow over Three-Dimensional Swept and Twisted RAE Wing*

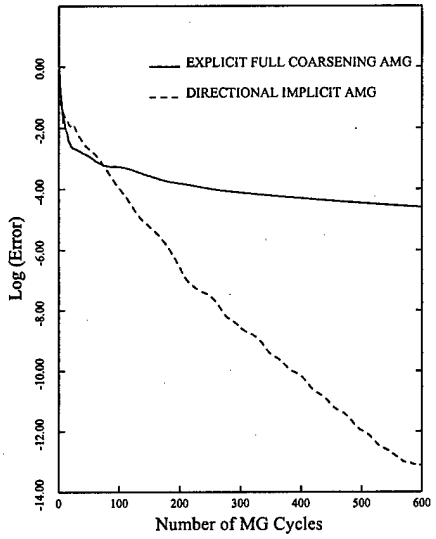

FIG. 5.2. *Comparison of Convergence Rate Achieved by Directional Implicit Agglomeration Multigrid versus Explicit Agglomeration Multigrid for Flow over Wing*

The scalability of the solver for these two cases is examined on an SGI Origin 2000 and a Cray T3E-600 machine. The SGI Origin 2000 machine contains 128 MIPS R10000 195 Mhz processors with 286 Mbytes of memory per processor, for an aggregate memory capacity of 36.6 Gbytes. The Cray T3E contains 512 DEC Alpha 300 Mhz processors with 128 Mbytes of memory per processor, for an aggregate memory capacity of 65 Gbytes. All the cases reported in this section were run in dedicated mode.

Figures 5.3 and 5.4 show the relative speedups achieved on the two hardware platforms for the RAE wing case, while Figures 5.5 and 5.6 depict the corresponding results for the ONERA M6 wing case. For the purposes of these figures, perfect speedups were assumed on the lowest number of processors for which each case was run, and all other speedups are computed relative to this value. In all cases, timings were measured for the single grid (non-multigrid) algorithm, the multigrid algorithm using a V-cycle, and the multigrid algorithm using a W-cycle. Note that the best numerical convergence rates are achieved using the W-cycle multigrid algorithm.

For the coarse RAE wing case, the results show good scalability up to moderate numbers of processors, while the finer ONERA M6 wing case shows good scalability up to the maximum number of processors on each machine, with only a slight drop-off at the higher numbers of processors. This is to be expected, since the relative ratio of computation to communication is higher for finer grids. This effect is also demonstrated by the superior scalability of the single grid algorithm versus the multigrid algorithms, and of the V-cycle multigrid algorithm over the W-cycle multigrid algorithm (i.e., the W-cycle multigrid algorithm performs additional coarse grid sweeps compared to the V-cycle algorithm). Note that for the RAE wing test case on 512 processors of the T3E, the fine grid contained only 348 vertices per processor, while the coarsest level contained a mere 13 points per processor. While the W-cycle algorithm suffers somewhat in computational

performance for coarser grids on high processor counts, the parallel performance of the W-cycle improves substantially for finer grids. Numerically the most robust and efficient convergence rates are achieved using this cycle. While these results reveal faster single processor computational rates for the Origin 2000, the Cray T3E-600 demonstrates higher scalability. In all cases, the fastest overall computational rates are achieved on the 512 processor configuration of the T3E-600.

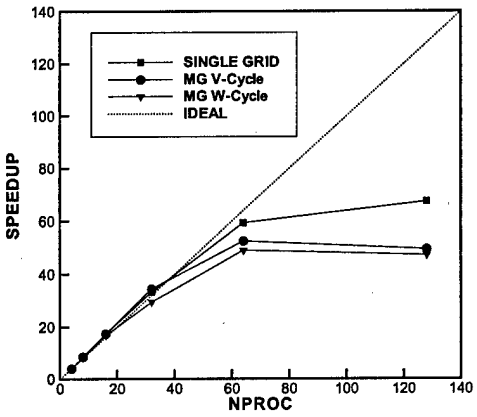

FIG. 5.3. *Observed Speedups for RAE Wing Case (177,837 grid points) on SGI Origin 2000*

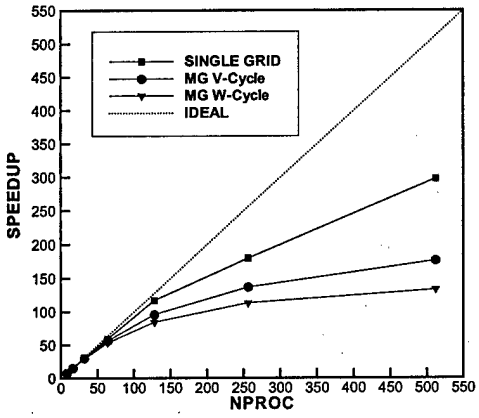

FIG. 5.4. *Observed Speedups for RAE Wing Case (177,837 grid points) on Cray T3E-600*

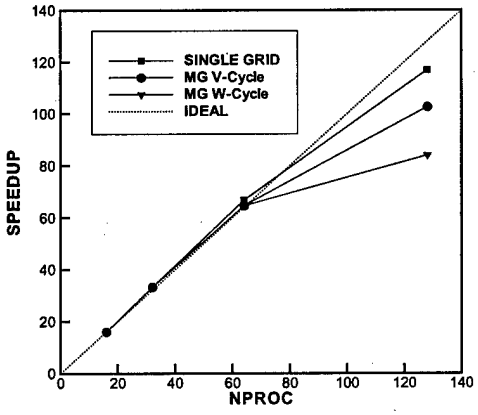

FIG. 5.5. *Observed Speedups for ONERA M6 Wing Case (1.98 million grid points) on SGI Origin 2000*

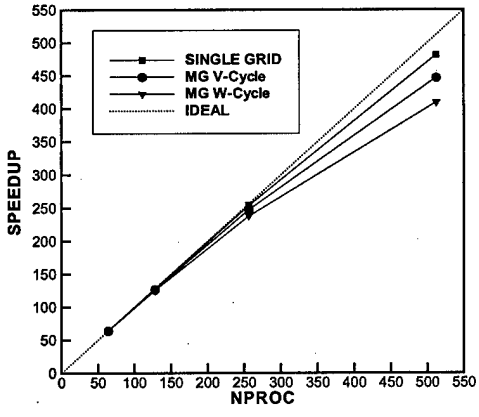

FIG. 5.6. *Observed Speedups for ONERA M6 Wing Case (1.98 million grid points) on Cray T3E-600*

6. Parallel Scalability of Multigrid. The previous results demonstrate that multigrid algorithms provide good scalability on large numbers of processors for reasonably sized problems. For a single-grid explicit scheme, the ratio of computation to communication remains relatively constant as the number of processors is increased, provided the problem size is increased proportionately. While this is also true for the finest grid levels of a multigrid algorithm, the coarsest grid of a multigrid algorithm must retain a fixed size as the fine grid problem size is increased (and extra grid levels are added). Thus, the parallel efficiency

of the coarsest grid of the multigrid algorithm will deteriorate continuously as the number of processors is increased, ultimately reaching a point where there are fewer coarse grid points than processors. This has lead to speculation in the past that multigrid methods should scale unfavorably for very large numbers of processors [2]. However, as the problem size is increased, the work on the coarsest grid becomes a smaller fraction of the overall work, and a simple argument can be made which suggests a well formulated multigrid algorithm will scale asymptotically to within a constant of its underlying fine grid smoothing algorithm.

If N denotes the number of fine grid points for a problem to be solved, and P denotes the number of processors, then the ratio $\frac{N}{P}$, i.e., the number of grid points per processor, is a measure of the computation work to be performed on the fine grid by each processor. Similarly, $\lceil\frac{N}{P}\rceil^{\frac{2}{3}}$ represents the surface area of this partition (in three dimensions), which is a measure of the communication to be performed by each processor on the fine grid. The ratio of computation to communication for the fine grid is therefore

$$\frac{N}{P} \div \lceil\frac{N}{P}\rceil^{\frac{2}{3}} = \lceil\frac{N}{P}\rceil^{\frac{1}{3}} \tag{6.1}$$

which is constant if N and P are increased proportionately to each other, as explained above.
For a multigrid V-cycle which performs one smoothing on each grid level, and where the coarsening factor between fine and coarse grids is 8, the total work per multigrid cycle for each processor is thus:

$$\frac{N}{P} \quad \times \lceil 1 + \frac{1}{8} + \frac{1}{64} + ... \rceil = \frac{8}{7}\frac{N}{P} \tag{6.2}$$

whereas the total communication per multigrid cycle is give by:

$$\lceil\frac{N}{P}\rceil^{2/3} \times \lceil 1 + \frac{1}{4} + \frac{1}{16} + ... \rceil = \frac{4}{3}\lceil\frac{N}{P}\rceil^{2/3} \tag{6.3}$$

so that the ratio of computation to communication for the entire multigrid cycle is given by:

$$\lceil\frac{P}{N}\rceil^{1/3} \times \frac{7}{6} \tag{6.4}$$

which is similar to that observed for the single grid algorithm to within a multiplicative constant. Therefore, in spite of the poor scalability of the fixed coarse grid problem size, the entire multigrid algorithm can be expected to scale similarly to the fine grid algorithm for increasing problem size on large numbers of processors. Although a W-multigrid cycle operating on 8:1 coarsened grid levels can also be shown to scale favorably, lower coarsening ratios (such as 4:1) will ultimately lead to worse asymptotic scalability and should be avoided. This current argument neglects the inter-grid communication, which is small in the current implementation, and non-existent when a fully nested fine-coarse grid partitioning strategy is employed.

7. Large Test Case Results. The next test case is intended to demonstrate the capability of running very large cases on large numbers of processors. The configuration involves the external flow over an aircraft with deployed flaps. The freestream Mach number is 0.2, and the Reynolds number is 1.6 million and the experimental flow incidence varies over a range of -4 degrees up to 24 degrees. The computations are all performed at zero yaw angle, and therefore only include one half of the symmetric aircraft geometry, delimited by a symmetry plane.

An initial grid of 3.1 million points was generated for this configuration using the VGRID unstructured tetrahedral grid generation package [18, 14]. A finer grid containing 24.7 million vertices was then obtained through h-refinement of the initial grid, i.e., by subdividing each cell of the initial grid into eight smaller self-similar cells. The refinement operation was performed sequentially on a single processor of an SGI Origin 2000, and required approximately 10 Gbytes of memory and 30 minutes of CPU time. Figure 7.1 depicts the surface grid for the initial 3.1 million point mesh in the vicinity of the flap system.

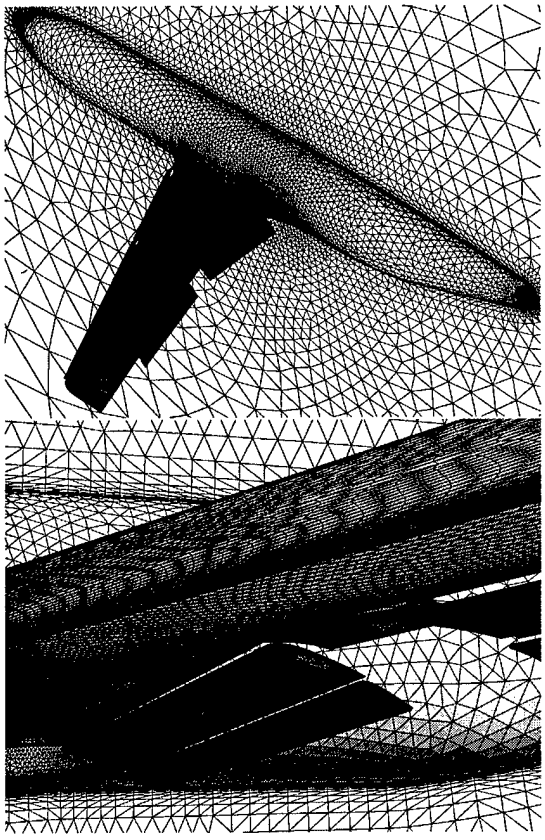

FIG. 7.1. *Illustration of Surface Grid for Initial 3.1 million Point Grid for Three-Dimensional High-Lift Configuration*

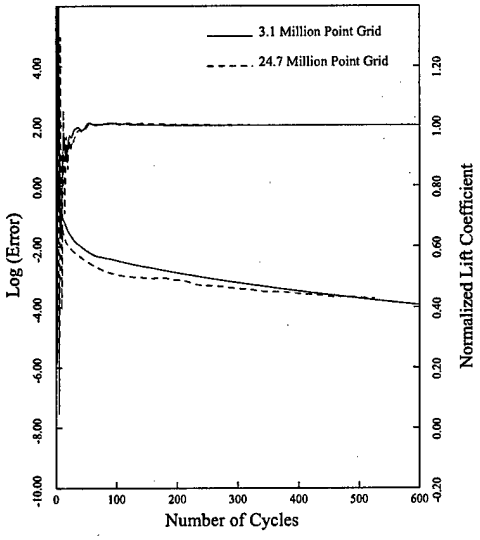

FIG. 7.2. *Convergence Rate for Coarse (3.1 million pt) Grid using 5 Multigrid Levels and Low Mach Number Preconditioning and Fine (24.7 million pt) Grid using 6 Multigrid Levels and no Preconditioning at 0.2 Mach Number and 10 degrees Incidence*

The convergence history obtained by the multigrid algorithm for the initial and refined grids at an incidence of 10 degrees is shown in Figure 7.2. In both cases, the line implicit algorithm is employed as a smoother, but the directional agglomeration strategy has been abandoned in favor of the simpler isotropic agglomeration strategy. Memory savings are realized from the faster (8:1) coarsening rates achieved by the isotropic agglomeration algorithm (as opposed to 4:1 for the directional algorithm), and from the preceding argument, the overall multigrid algorithm can be expected to scale asymptotically to within a constant of the single grid algorithm. On the other hand, as a result of the isotropic agglomeration procedure, the convergence rates for these cases are slower than those observed in the previous two cases. However, the multigrid algorithm still delivers convergence rates which are relatively insensitive to the overall grid resolution, as demonstrated by the results of Figure 7.2.

The 3.1 million point grid case has been run on a variety of machines. The scalability of this case on

the Cray T3E and SGI Origin 2000 is similar to that illustrated in Figures 5.5 and 5.6. This case requires a total of 7 Gbytes of memory and 80 minutes on 128 processors (250 MHz) of the Origin 2000, or 62 minutes on 256 processors of the Cray T3E-600 for a 500 multigrid cycle run.

The scalability of the single grid and multigrid algorithms for this case on the IBM-based ASCI Blue Pacific machine, and the Intel-based ASCI Red machine is depicted in Figures 7.3 and 7.4. The ASCI Blue Pacific machine, located at Lawrence Livermore National Laboratory in California, consists of 320 nodes, with each node containing 4 IBM 332Mhz 604e shared memory processors. The results in Figure 7.3 employed all four processors at each requested node, which was found to incur a 20% timing penalty over an approach which employed only a single processor per node, using four times as many nodes. This penalty is likely due to the fact that the available node bandwidth must be shared between the four processors in this node, but this approach is necessary for accessing large numbers of processors. In all cases, a purely MPI-based implementation has been used. Good scalability is obtained up to approximately 256 processors, after which the parallel efficiency begins to drop off. This is partly due to the relatively small size of the problem for this number of processors.

The ASCI Red machine, located at Sandia National Laboratory in New Mexico, contains up to 4500 dual cpu nodes. The individual CPUs consist of 333 Mhz Intel Pentium Pro processors. The results in Figure 7.4 only made use of a single cpu per node, since there is no way to access both processors on a node with a purely MPI-based code. On this machine, good scalability is observed up to 2048 processors for the single grid case, and up to 1024 processors for the multigrid case. The multigrid case would likely scale well at 2048 processors, although such a run has not been performed to date.

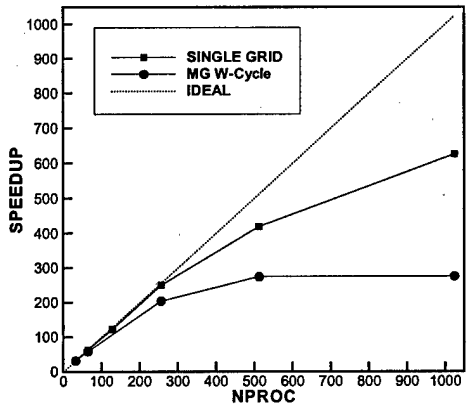

FIG. 7.3. *Observed Speedups for 3.1 million point grid aircraft case on ASCI Blue Pacific Machine*

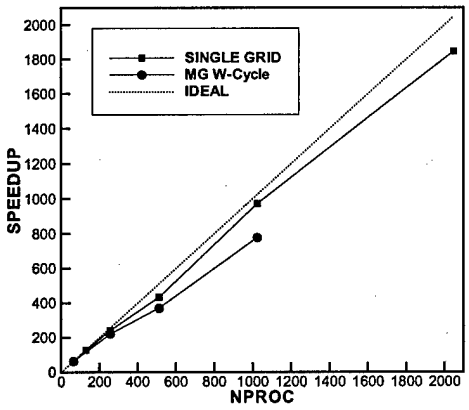

FIG. 7.4. *Observed Speedups for 3.1 million point grid aircraft case on ASCI Red Machine*

The 24.7 million point case was run on the Cray T3E-600 machine using 512 processors. This case requires 52 Gbytes of memory, and 4.5 hours for 500 multigrid cycles, which includes 30 minutes of I/O time to read the grid file (9 Gbytes), and write the solution file (2 Gbytes). The fine grid case was also benchmarked on a larger Cray-T3E-1200E machine. The Cray T3E-1200E contains 600 MHz DEC Alpha processors as well as an upgraded communication chip, as compared to the previously mentioned T3E-600 (300MHz processors). This particular machine contained 1520 processors each with a minimum of 256 Mbytes per processor. Figure 7.5 depicts the speedups obtained by the single grid, and the five level and six level multigrid runs on the

24.7 million point grid running on 256, 512, 1024 and 1450 processors. The single grid computations achieve almost prefect scalability up to 1450 processors, while the speedups achieved by the multigrid runs are only slightly below the ideal values. The six level multigrid case could not be run on the maximum number of processors, since the partitioning of the coarsest level resulted in empty processors with no grid points. While this does not represent a fundamental problem, the software was not designed for such situations. In any case, the five level multigrid runs are the most efficient overall, since there is little observed difference in the convergence rate between the five and six level multigrid runs. The single grid results are included simply for comparison with the multigrid algorithm, and are not used for actual computations since convergence is extremely slow. The computation times are depicted in Table 7.1. On 512 processors of the Cray T3E-1200E, the 5 level multigrid case requires 19.7 seconds per cycle, as compared to 28.1 seconds per cycle on the 512 processor Cray T3E-600, which corresponds to an increase in speed of over 40% simply due to the faster individual processors. On 1450 processors, the same case required 7.54 seconds per cycle, or 63 minutes of computation for a 500 multigrid cycle run. A complete run required 92 minutes, which includes 29 minutes of I/O time, although no attempt at optimizing I/O was made.

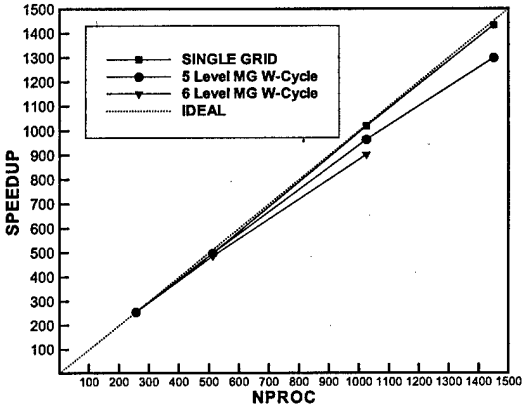

FIG. 7.5. *Observed Speedups for 24.7 million point Grid Case on 1520 Processor Cray T3E-1200E*

TABLE 7.1

Timings and Estimated Computational Rates for 24.7 million point Grid Case on Various Cray T3E Configurations; Computational Rates are obtained by linear scaling according to wall clock time with smaller problems run on the Cray C90 using the hardware performance monitor for Mflop ratings

24.7 Million Pt Case (5 Multigrid Levels)			
Platform	Procs	Time/Cyc	Gflop/s
T3E-600	512	28.1	22.0
T3E-1200E	256	38.3	16.1
T3E-1200E	512	19.7	31.4
T3E-1200E	1024	10.1	61.0
T3E-1200E	1450	7.54	82.0

8. Conclusions. While the calculations described in this paper have demonstrated good overall performance for large problems on thousands of processors, these calculations are limited by the pre-processing operations such as grid partitioning, and coarse multigrid level construction, which are currently performed sequentially on a single processor using shared memory machines. The parallelization of these operations for distributed memory computer architectures is required before much larger calculations can be attempted. I/O issues, including bandwidth, file size, and file transfer between machines is also a serious issue for calculations of this size, which must be addressed in future work.

While a distributed memory approach has been adopted, there is a growing trend to employ clusters of mid-sized shared memory machines. This is perhaps due to the fact that the most marketable scientific computer architectures are mid-sized shared memory machines. Clustering such machines together is seen as a cost effective approach to building customized very high performance systems. An effective programming

model for such architectures may involve the use of mixed shared memory (using OpenMP) and distributed memory (using MPI) libraries. An extension of the current solver to a mixed shared-distributed memory model is currently under way.

9. Acknowledgements. Special thanks are due to David Whitaker and Cray Research for dedicated computer time, and to S. Pirzadeh for his grid generation expertise. This work was partly funded by an Accelerated Strategic Computing Initiative (ASCI) Level 2 grant from the US Department of Energy.

REFERENCES

[1] *Special course on parallel computing in CFD*, May 1995. AGARD Report-807.

[2] P. O. FREDRICKSON AND O. A. MCBRYAN, *Parallel superconvergent multigrid*, in Multigrid Methods, Theory, Applications, and Supercomputing, S. McCormick, ed., vol. 110, Marcel Dekker, New York, 1988, pp. 195–210.

[3] W. GROPP, E. LUSK, AND A. SKJELLUM, *Using MPI: Portable Parallel Programming with the Message Passing Interface*, MIT Press, Cambridge, MA, 1994.

[4] B. HENDRICKSON AND R. LELAND, *The Chaco user's guide: Version 2.0.* Tech. Rep. SAND94-2692, Sandia National Laboratories, Albuquerque, NM, July 1995.

[5] ——, *A multilevel algorithm for partitioning graphs*, in Proceedings Proc. Supercomputing '95, ACM, Dec. 1995.

[6] G. KARYPIS AND V. KUMAR, *A fast and high quality multilevel scheme for partitioning irregular graphs*, Tech. Report 95-035, University of Minnesota, 1995. A short version appears in Intl. Conf. on Parallel Processing 1995.

[7] ——, *A fast and high quality multilevel scheme for partitioning irregular graphs.* To appear in SIAM J. Sci. Comput., 1998.

[8] D. E. KEYES, D. K. KAUSHIK, AND B. F. SMITH, *Prospects for CFD on petaflops systems*, in CFD Review M. Hafez, et al., eds., Wiley, New York, 1997.

[9] M. LALLEMAND, H. STEVE, AND A. DERVIEUX, *Unstructured multigridding by volume agglomeration: Current status*, Computers and Fluids, 21 (1992), pp. 397–433.

[10] D. J. MAVRIPLIS, *Multigrid strategies for viscous flow solvers on anisotrpic unstructured meshes*, in Proceedings of the 13th AIAA CFD Conference, Snowmass, CO, June 1997, pp. 659–675. AIAA Paper 97-1952-CP.

[11] ——, *Directional agglomeration multigrid techniques for high-Reynolds number viscous flows.* AIAA Paper 98-0612, Jan. 1998.

[12] ——, *On convergence acceleration techniques for unstructured meshes.* AIAA Paper 98-2966, presented at the 29th AIAA Fluid Dynamics Conference, Albuquerque, NM, June 1998.

[13] D. J. MAVRIPLIS, R. DAS, J. SALTZ, AND R. E. VERMELAND, *Implementation of a parallel unstructured Euler solver on shared and distributed memory machines*, The J. of Supercomputing, 8 (1995), pp. 329–344.

[14] D. J. MAVRIPLIS AND S. PIRZADEH, *Large-scale parallel unstructured mesh computations for 3D high-lift analysis.* AIAA Paper 99-0537, presented at the 37th AIAA Aerospace Sciences Meeting, Reno, NV, Jan. 1999.

[15] D. J. MAVRIPLIS AND V. VENKATAKRISHNAN, *A unified multigrid solver for the Navier-Stokes equations on mixed element meshes*, International Journal for Computational Fluid Dynamics, 8 (1997),

pp. 247–263.

[16] E. MORANO AND A. DERVIEUX, *Looking for O(N) Navier-Stokes solutions on non-structured meshes*, in 6th Copper Mountain Conf. on Multigrid Methods, 1993, pp. 449–464. NASA Conference Publication 3224.

[17] N. PIERCE AND M. GILES, *Preconditioning on stretched meshes.* AIAA Paper 96-0889, Jan. 1996.

[18] S. PIRZADEH, *Viscous unstructured three-dimensional grids by the Advancing-Layers method.* AIAA Paper 94-0417, Jan. 1994.

[19] P. L. ROE, *Approximate Riemann solvers, parameter vectors and difference schemes*, J. Comp. Phys., 43 (1981), pp. 357–372.

[20] P. R. SPALART AND S. R. ALLMARAS, *A one-equation turbulence model for aerodynamic flows*, La Recherche Aérospatiale, 1 (1994), pp. 5–21.

[21] B. V. E. T. C. H. TAI AND L. MESAROS, *Local preconditioning in a stagnation point*, in Proceedings of the 12th AIAA CFD Conference, San Diego, CA, June 1995, pp. 88–101. AIAA Paper 95-1654-CP.

[22] E. TURKEL, *Preconditioning methods for solving the incompressible and low speed compressible equations*, J. Comp. Phys., 72 (1987), pp. 277–298.

[23] ——, *Preconditioning-squared methods for multidimensional aerodynamics*, in Proceedings of the 13th AIAA CFD Conference, Snowmass, CO, June 1997, pp. 856–866. AIAA Paper 97-2025-CP.

[24] B. VAN LEER, C. H. TAI, AND K. G. POWELL, *Design of optimally-smoothing multi-stage schemes for the Euler equations.* AIAA Paper 89-1933, June 1989.

[25] V. VENKATAKRISHNAN, *Implicit schemes and parallel computing in unstructured grid CFD*, in VKI Lecture Series VKI-LS 1995-02, Mar. 1995.

[26] J. M. WEISS AND W. A. SMITH, *Preconditioning applied to variable and constant density time-accurate flows on unstructured meshes.* AIAA Paper 94-2209, June 1994.